UNIABUJA Journal of Engineering and Technology
https://ujet.uniabuja.edu.ng/
ISSN: 2714-3236 (Online); 2714-3228 (Print)
Volume 2, Issue 2, 2025; 368-376

Enhancing Web Application Security Through Automated SQL Injection Detection Using Neural Networks

Amina I. ABUBAKAR[1*], Hashim I. BISALLAH[2], Khadijat M. KABIR[3]

[1*,2,3]*Department of Computer Science, University of Abuja, Abuja, Nigeria*

[1*]amina.imam@uniabuja.edu.ng, [2]hashim.bisallah@uniabuja.edu.ng, [3]khadijat.mohammed2017@uniabuja.edu.ng

Abstract
SQL injection (SQLi) attacks are one of the major threats to web application security, especially on e-commerce platforms. These attacks exploits the weaknesses in user input which enables attackers to have access and manipulate database queries to compromise data integrity. This study aims to develop an automated SQL injection detection system using Neural Networks to improve on the security of web applications. A labeled dataset of SQL injection patterns was created to train three machine learning models namely: Naive Bayes, Random Forest, and Deep Neural Network. The models evaluation was done using accuracy, precision, specificity, and F1 score. The results shows that the Neural Network model outperformed the two others by achieving an accuracy of 99.1%, a precision of 94.2%, a specificity of 98.1%, and a F1 score of 0.961. These results shows that Neural Networks is efficient in detecting SQL injection attacks. Finally, this study provides a comparative insights to earlier research by exploring different potential deployment scenarios, and identifies avenues for future work.

Keywords— *SQL Injection, Web Application Security, Neural Networks, Machine Learning, Deep Learning, E-commerce Security.*

1.0 Introduction

The internet and web applications plays an important role in modern business operations, many organizations rely on them to manage their sensitive data and transactions. E-commerce platforms stores valuable user information such as usernames, passwords, and banking details. The most essential of these systems are relational databases which operates through Structured Query Language (SQL). These weaknesses are common in SQL and web applications are exploited by malicious actors to manipulate confidential information that poses serious threats to businesses (Farooq, 2021).

One of the most common threats is the SQL Injection (SQLi) which enable attackers to bypass the authentication process, retrieve sensitive data, and compromise entire systems. SQLi is ranked among the top ten web security threats in the Open Web Application Security Project (OWASP) list for over a decade. High-profile attacks like the 2002 attack on Guess.com which highlights the devastating impact of such breaches. Also, the recent statistics from Statista (2020) reveals an average of over 953,000 blocked web attacks daily which underscores the growing scale of this problem. SQL injection attackers injects malicious SQL code into the input fields to manipulate the backend queries and executes arbitrary commands. These attacks compromise data integrity, violate user privacy, and lead to regulatory penalties. Despite the deployment of traditional security measures which are the static and dynamic analysis, rule-based filtering, and blacklisting, these techniques still fall short due to evolving methods of attacks and also the emergence of novel payloads (Rankothge et al., 2020).

Some progress has been made using machine learning but there exist challenges such as manual feature extraction, overfitting, and poor generalization. Conventional models has failed to exploit the syntax and context of SQL queries. There is also a need for an automated solutions that is capable of learning complex patterns and adapting to emerging new threats (Singh et al., 2016). This study proposes the use of deep neural networks (DNNs) for automated SQL injection detection in e-commerce web applications by leveraging on the strengths of deep learning and its capacity for automatic feature extraction and high accuracy.

2.0 Background and Related Work

The combination of detection and prevention techniques are required to reduce SQL injection (SQLi) attacks in modern web applications. These approaches ranges from traditional rule-based mechanisms to advanced machine learning and neural network-based models. This section outlines some of the methods commonly used in the field.

Received: 26-05-2025 / Accepted: 25-06-2025 / Published: 28-09-2025

https://doi.org/10.70118/ujet.2025.0202.34

2.1 Traditional SQL Injection Detection Methods

Early methods of detecting SQLi attacks relied heavily on signature-based and rule-based systems. Tools like Snort and Web Application Firewalls (WAFs) inspect the network traffic and compare it against all ready defined patterns or rules (Inyong et al. 2012). These systems are effective against known form of attack vectors but they find it difficult with obfuscated or zero-day SQLi variants thereby exhibiting poor adaptability and high false positive/negative rates.

Regular expression matching and input sanitization are among the primary preventive techniques (Halfond, et al. 2006). These methods require constant updates and are sometimes bypassed by polymorphic or encoded SQL payloads.

2.2 Machine Learning-Based Approaches

Researchers have introduced several machine learning (ML) models for SQLi detection to overcome the problems of static rule-based systems. ML models like Support Vector Machines (SVMs), Decision Trees, Random Forests, and Naïve Bayes have demonstrated improved generalization and adaptability (Hasan et al. 2019).

Sheykhkanloo (2017) applied decision trees and achieved significant accuracy in SQLi queries identification from a labeled dataset. ML models leverage features such as query length, character frequency, presence of SQL keywords, and entropy values. The performance of ML approaches depends on the quality of feature engineering and may weaken when faced with novel or slightly modified attacks.

2.3 Neural Network-Based Detection

Deep learning models such as Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Convolutional Neural Networks (CNNs), have gained traction in SQLi detection due to their ability to automatically learn hierarchical patterns from raw input data.

Dasari et al. (2025) applied VAE, CWGAN-GP, and U-Net for synthetic SQL query generation, enhancing detection accuracy and adaptability, though training was resource-intensive. Arasteh et al. (2024) used binary GWO for feature selection with traditional ML, improving SQLi detection accuracy but with high computational cost.

Zhang et al. (2019) proposed an LSTM-based model that analyzes SQL queries as sequential data by capturing contextual dependencies successfully between tokens. Their model performed better than the traditional ML classifiers in detecting both known and obfuscated attacks.

Alwan et al. (2020) developed a CNN-based classifier that treats SQL queries as sequences of characters by enabling automatic feature extraction without domain-specific knowledge. These approaches reduced false positives while still maintaining high detection accuracy.

The challenges of Neural network models is that it requires a large annotated datasets and are often viewed as black-box systems thereby making interpretation and real-time deployment more complex. Therefore, hybrid approach which combines ML and DL have been proposed to balance interpretability and performance (Tian et al., 2020).

2.4 Comparative Analysis

Most of the comparative studies reveals that deep learning-based models performed better traditional and machine learning-based models in SQLi detection tasks. The advantages is in their ability to learn complex, nonlinear relationships without manual feature engineering. However, issues such as training cost, model interpretability, and data imbalance still need to be addressed.

Furthermore, research has explored adversarial SQLi inputs that can bypass deep learning models, prompting the integration of adversarial training and explainable AI (XAI) techniques to improve robustness and transparency (Alenezi et al., 2021).

2.5 Research Gap and Motivation

Despite the various progress made in the detection of SQLi, a significant gap exists in creating scalable, real-time, and explainable neural network-based models which are personalized to evolving web applications in the e-commerce sector. Most of the existing models are not optimized for speed and their evaluations are done on an outdated or synthetically generated datasets. This paper aims to bridge this gap by proposing a robust deep learning-based detection framework which leverage on real-world SQLi datasets thereby focusing on both accuracy and deployment feasibility.

3.0 Methodology

This study proposes a deep learning-based framework for the detection of SQL Injection Attacks (SQLIAs) in web applications by leveraging the neural networks and a well-structured feature engineering. The main

Received: 26-05-2025 / Accepted: 25-06-2025 / Published: 28-09-2025

https://doi.org/10.70118/ujet.2025.0202.34

aim of the detection system is to differentiate between destructive SQL injection statements and benign, non-malicious statements which includes both ordinary SQL commands and general user input text. The framework is structured to automate the classification process by moving away from traditional manual rule design by employing a novel string kernel approach that transforms input queries into high-dimensional mathematical representations suitable for machine learning models. The overall architecture of machine leraning framework is illustrated in Figure 1.

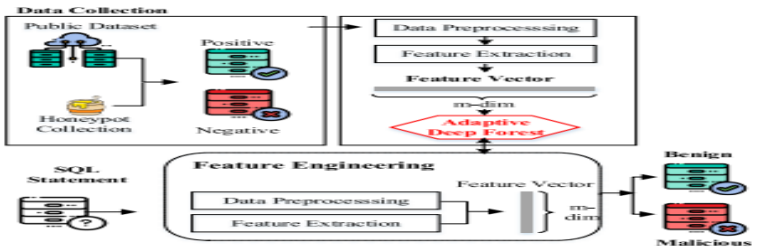

Figure 1: Machine Learning Framework of Detecting SQL Injection Attacks

The dataset used in this study was gotten from Kaggle which is an open source machine learning repository. It comprises of 30,919 data entries that meets the requirements of this research. Each SQL query is first passed through a tokenizer that convert the input into tokens to represents the relevant features. These tokens are labeled as either malicious or non-malicious and subsequently fed into the classifier for final classification. This design aims to identify and mitigate queries that could bypass authentication, alter, or delete the underlying database.

The data is prepared for model training by executing a comprehensive feature engineering phase which includes feature extraction and feature selection. The "Blank Separation Method" was employed to extract terms based on spaces and to ensure a granular view of query components. The SQL injection statements were examined for tokens and keywords such as select, insert, drop, union, and symbols like ;, --, and =, which are frequently used to manipulate SQL queries.

Data preparation played an important role to ensure model accuracy and generalization. It handled missing values, data cleansing, normalization, and noise reduction. Noisy data were smoothed and outliers were removed to prevent overfitting. Domain knowledge was used to identify inconsistencies within the dataset while scatter plots were used to explore non-linear relationships. The data preprocessing phase are automatic feature extraction, normalization, and segmentation. Tokenization was done using the word-pause method which followed by filtering via the TF-IDF algorithm to retain only significant terms. Tokens with over 80% frequency or fewer than two occurrences were excluded as they provided little to no discriminatory power for classification.

Three regularization like the Dropout, data regularization (L1 or L2), and Early Stopping was implemented to improve the robustness of the model and also to reduce the likelihood of overfitting. Dropout deactivates neurons during training forcing the model to generalize better. The neural network contains three hidden layers where neurons were selectively dropped during training with specified probabilities and improved model generalization. During the test phase, all neuron connections are returned to increase performance. Early Stopping was used to terminate training when the validation loss ceased to prevent over-training. Figure 2 shows the network architecture for the neural network.

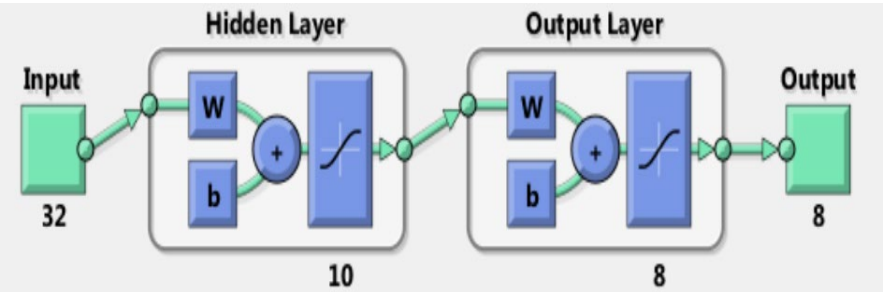

Figure 2: Network Architecture for the Neural Network Component of the Model

Since SQLIA detection is just a binary classification problem, the classification task was done by deep neural network and the dataset was labeled such that SQL injection queries were marked as 1 (malicious) and non-SQL injection queries marked as 0 (non-malicious). Non-malicious queries included both benign SQL and non-SQL text statements. The neural network architecture used in the study contains multiple hidden layers with each layer transforming the input data through weighted computations and bias adjustments. Rectified

Received: 26-05-2025 / Accepted: 25-06-2025 / Published: 28-09-2025

https://doi.org/10.70118/ujet.2025.0202.34

Linear Unit (ReLU) function was used to handle the activation and it was chosen for its computational efficiency and ability to handle non-linear transformations. Model training involved forward propagation and back propagation using the Adam optimization algorithm.

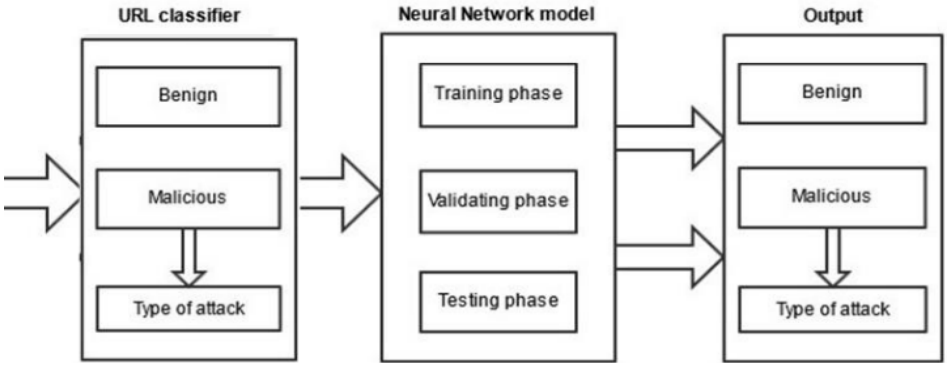

Figure 3: Components of the proposed neural network-based model for detection and classification of SQLi attacks

The performance of the proposed model was evaluated using metrics such as Accuracy, Precision, Recall, F1-Score, and the ROC curve. The confusion matrix checked the model efficiency in terms of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Accuracy and precision were particularly emphasized, as they directly reflect the model's ability to correctly classify both SQL injection and benign queries. The ROC curve further helped in assessing the model's discriminative power across different threshold settings.

All algorithms were implemented using Python with the support of Keras and TensorFlow libraries. Development was carried out by Google Colab, and online Jupyter Notebook environment. The computational hardware used included a system with an Intel Core i7-9900HQ CPU and 16GB of RAM, providing sufficient processing power for training deep neural networks. The summary table for the methodology is shown in Table 1.

Table 1: Summary table for methodology

Stage	Method/Technique
Dataset Collection	SQLi dataset from Kaggle
Data Preparation	Data cleaning, handling missing values, noise smoothing, normalization
Feature Engineering	Blank Separation Method, feature selection
Data Preprocessing	Tokenization (Word Pause), TF-IDF, filtering rare/common tokens
Model Architecture	Deep Neural Network (3 hidden layers, ReLU activation)
Overfitting Prevention	Dropout, L1/L2 Regularization, Early Stopping
Optimizer	Adam (Adaptive Moment Estimation)
Training Algorithm	Backpropagation
Classification	Binary classification (SQLi = 1, Non-SQLi = 0)
Evaluation Metrics	Accuracy, Precision, Recall, F1-Score, ROC Curve, Confusion Matrix
Development Tools	Python, TensorFlow, Keras, Google Colab, Jupyter Notebook

4.0 Evaluation Metrics

The proposed SQL injection detection model was evaluated using several standard classification metrics. The metrics are Accuracy, Precision, Recall, F1-Score, and the Receiver Operating Characteristic (ROC) Curve. The confusion matrix was used as the primary tool to outline the different combinations of predicted and actual classification results. The confusion matrix categorized the predictions into True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). True Positives are the malicious SQL injection statements that were correctly identified, while True Negatives are the non-malicious statements that were accurately classified. False Positives occurs when normal SQL statements are incorrectly labeled as malicious, and False Negatives are malicious statements misclassified as benign.

Accuracy which is one of the key performance metrics is defined as the ratio of correctly classified instances (TP and TN) to the total number of instances. Precision measured the proportion of correctly predicted positive observations to the total predicted positives. Recall evaluated the model's ability to identify actual

Received: 26-05-2025 / Accepted: 25-06-2025 / Published: 28-09-2025

https://doi.org/10.70118/ujet.2025.0202.34

malicious injections correctly. The F1-Score provided a balanced measure of a model's performance when there is an imbalance between classes.

The ROC Curve assessed the trade-off between the True Positive Rate and the False Positive Rate at various threshold settings. A model with a ROC value closer to 1 is considered ideal, while values near 0 indicate poor performance. These metrics collectively provided a comprehensive view of the classifier's effectiveness. A higher proportion of True Positives and True Negatives in the confusion matrix leads to improved Accuracy, Precision, Recall, and F1-Score values which are indicative of robust classification performance.

5.0 Experimental Results

This study investigates the effectiveness of three machine learning models which are Naïve Bayes, Random Forest, and Neural Networks for the detection of SQL injection attacks using a dataset that contained 30,919 labeled SQL queries. The performance of each model was evaluated using four key metrics: accuracy, precision, specificity, and F1 score, providing a comprehensive understanding of each model's strengths and limitations. The dataset overview is shown in Figure 4.

	Sentence	Label	Unnamed: 2	Unnamed: 3
0	" or pg_sleep (__TIME__) --	1	NaN	NaN
1	create user name identified by pass123 tempora...	NaN	1	NaN
2	AND 1 = utl_inaddr.get_host_address (...	1	NaN	NaN
3	select * from users where id = '1' or @ @1 ...	1	NaN	NaN
4	select * from users where id = 1 or 1#" (...	1	NaN	NaN
5	select name from syscolumns where id = ...	1	NaN	NaN
6	select * from users where id = 1 +$+ or 1 =...	1	NaN	NaN
7	1; (load_file (char (47,101,116,99,47...	1	NaN	NaN
8	select * from users where id = '1' or ||/1 ...	1	NaN	NaN
9	select * from users where id = '1' or \.<\ ...	1	NaN	NaN

Figure 4: Dataset

The Naïve Bayes model achieved an accuracy of 82% which identifies a successful large portion of legitimate and malicious queries. However, its precision was 68% that indicates a relatively higher rate of false positives where benign queries were mistakenly flagged as attacks. While the model's specificity was 72% which shows moderate effectiveness in distinguishing safe queries, the F1 score of 0.805 revealed a balance between identifying threats and avoiding false alarms. Despite its simplicity and speed, the performance suggests that Naïve Bayes is not ideal for real-time security systems that require higher accuracy and fewer false positives. The SQL Injection Payload Categories is shown in Figure 5.

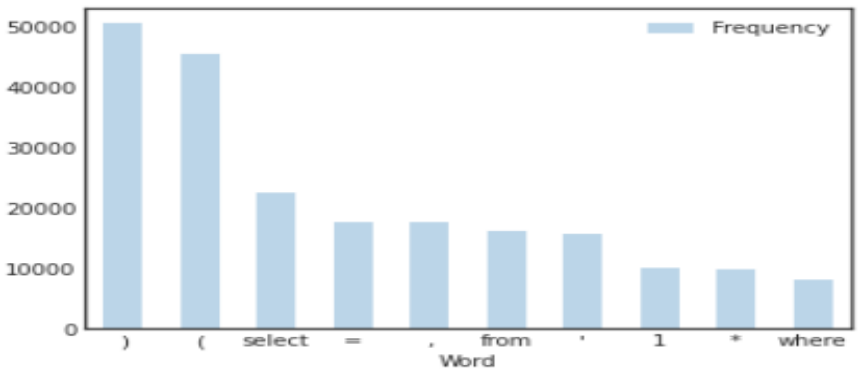

Figure 5: Bar chart of SQL Injection Payload Categories

Interaction refers to how a user's input and a web application's response are exchanged which reveals malicious behavior like SQL injection. Detection systems monitor these interactions for abnormal patterns such as unexpected SQL keywords or database error messages. Figures 6 and 7 illustrates the request sequences flagged as benign or malicious. To improve detection, feature engineering was used to refine data, and data augmentation was applied to handle the high variability of input queries thereby enhancing the model's ability to generalize.

Received: 26-05-2025 / Accepted: 25-06-2025 / Published: 28-09-2025

https://doi.org/10.70118/ujet.2025.0202.34

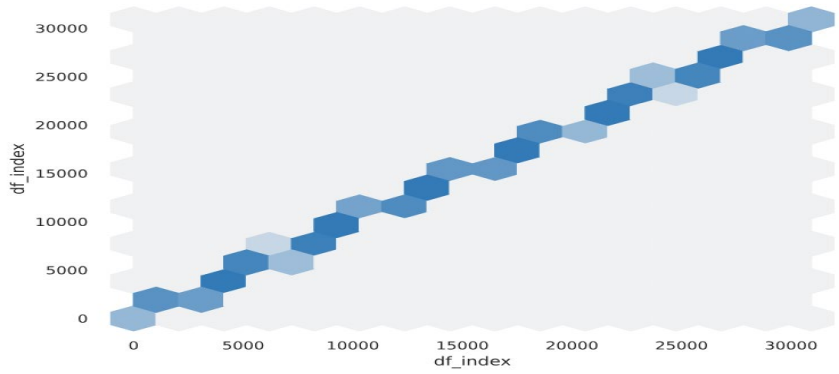

Figure 6: The hexbin plot

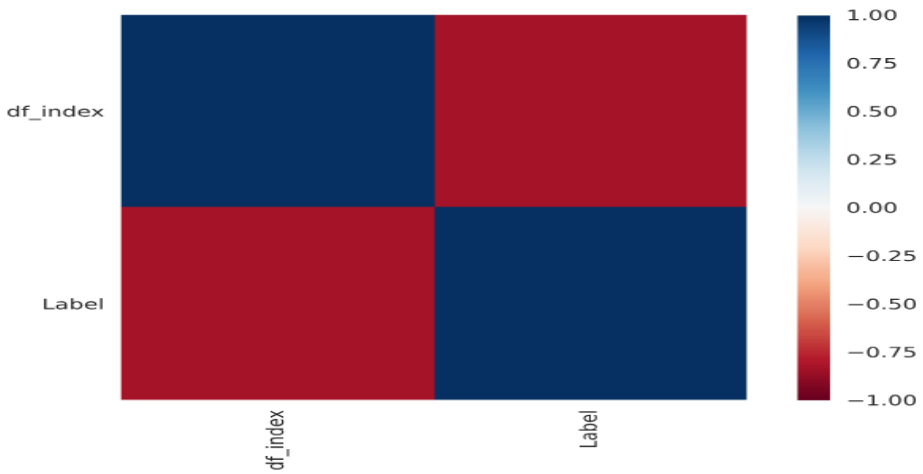

Figure 7: The correlation heatmap

In comparison, the Random Forest model demonstrated stronger performance with an accuracy of 91.1%. It classified more instances correctly than Naïve Bayes and improved in specificity (82.1%) that suggest a better ability to recognize legitimate queries and reduce false alerts. However, its precision was at 74.3% which indicates that there is a need for improvement to minimize misclassifications. The model's F1 score of 0.871 reflects a more robust and reliable detection capability thereby making it a better option for deployment in more demanding security environments.

Table 2: Model Performance Comparison

	Accuracy	**Precision**	**Specificity**	**Fi Score**
Naïve Bayes	0.820	0.680	0.720	0.805
Random Forest	0.911	0.743	0.821	0.871
Neural Network	0.991	0.942	98.1	0.961

The Neural Network model is the most effective among the three achieving an accuracy of 99.1% and a precision of (94.2%) which confirmed that the majority of the flagged queries were actually malicious. The specificity of 98.1% confirmed its ability to correctly classify legitimate traffic thereby reducing the chances of disruptions caused by false alarms. Additionally, the model achieved an F1 score of 0.961 which indicates a strong balance between precision and recall. The training of the model was efficient, taking only 0.4 seconds over 20 epochs. The neural network architecture was composed of three dense layers with 128, 64, and 1 neuron respectively by using batch normalization and dropout layers to enhance generalization and mitigate overfitting. A learning rate of 0.001 was optimized through hyperparameter tuning.

Table 3: Comparison with previous work

	Accuracy	**Fi Score**
This Study	0.991	0.961
Tang et al. (2020)	0.99	0.97

Received: 26-05-2025 / Accepted: 25-06-2025 / Published: 28-09-2025

https://doi.org/10.70118/ujet.2025.0202.34

The neural network's superiority is validated through a comparative analysis with related work by Tang et al. (2020), whose model achieved a similar accuracy (0.99) but a slightly lower F1 score (0.97). This comparison highlight the effectiveness of the proposed neural network approach in achieving a better balance between precision and recall. These results are also consistent with findings from Falor et al. (2021) and Lu et al. (2021) that demonstrated high accuracy in SQL injection detection using Convolutional Neural Networks (CNNs). While other studies, such as Triloka et al. (2022), reported even higher accuracy using SVMs, the current neural network model offers an excellent trade-off between performance and computational efficiency, which is essential for real-time intrusion detection.

6.0 Discussion

The aim of this study was to develop a machine learning-driven model to detect and prevent SQL injection (SQLi) attacks targeting e-commerce web applications. The research aimed to identify common SQLi vulnerabilities, build a labeled dataset of attack patterns, develop deep neural network models with overfitting reduction techniques, and compare their performance against existing approaches. The study evaluated three machine learning classifiers which are Naïve Bayes, Random Forest, and Neural Networks, using four performance metrics of accuracy, precision, specificity, and F1 score.

The Neural Network has the best performance by achieving an accuracy of 99.1%, precision of 94.2%, specificity of 98.1%, and an F1 score of 0.961. These metrics suggest that the model was able to classify almost all test queries correctly and also maintain a low rate of false positives and false negatives. The low training loss of 0.027 also confirms the model's effective learning without significant overfitting.

The advantage of the Neural Network model was evident when compared to Naïve Bayes and Random Forest classifiers. Random Forest performed well with accuracy of 91.1% and F1 score of 0.871, Naïve Bayes has an accuracy of 82% and lower precision. This supports the understanding that simpler probabilistic models like Naïve Bayes struggles to capture complex query structures in SQLi detection tasks. Moreover, the findings aligned with the results of Adebiyi et al. (2021), who also reported moderate performance for Naïve Bayes in similar contexts.

Comparative analysis with previous work validates the robustness of the proposed model. For instance, the model by Tang et al. (2020) achieved a similar accuracy of 99% and an F1 score of 0.97. The Neural Network model in this study achieved slightly higher F1 score of 0.961 which indicates a better balance between precision and recall. This enhancement could be attributed to the preprocessing strategies and regularization techniques used to improve generalization.

This study significantly contributes to the body of knowledge and supports the application of neural networks for SQL injection detection. The model's high accuracy, strong precision, and superior specificity confirm its utility in real-time intrusion detection systems. Nevertheless, the research is not without limitations. Recall data was not explicitly reported, and the dataset, while comprehensive, may benefit from further diversification. Future work should explore the generalizability of the model across larger and more heterogeneous datasets and examine the potential of hybrid detection mechanisms that combines machine learning with simulation and runtime techniques for holistic security solutions.

7.0 Conclusion and Future Work

SQL injection attacks continue to pose a significant threat to web applications, particularly in the e-commerce sector, where the consequences of successful intrusions can be severe. This study demonstrated the effectiveness of Neural Networks in detecting and mitigating such threats by developing a deep learning model that significantly outperformed traditional machine learning methods like Naive Bayes and Random Forest. With a near-perfect accuracy of 99.1%, high precision (94.2%), and specificity (98.1%), the Neural Network model showed a strong ability to distinguish malicious queries from legitimate ones while minimizing false positives. These results align with existing research that supports the use of advanced deep learning techniques for cybersecurity applications. Furthermore, the study contributed valuable insights into data preprocessing and potential deployment scenarios, such as integration into domain registration and web hosting platforms. Overall, the research highlights the viability of Neural Networks for real-time SQL injection detection and provides a solid benchmark for future innovations in this domain.

Although the current findings underscore the potential of Neural Networks for SQL injection detection, future research is needed to enhance and expand upon this work. This includes evaluating the model's performance on larger and more diverse datasets to improve generalizability and resilience against evolving attack patterns. Additionally, improving model interpretability will be crucial for building user trust and facilitating further refinement. Exploring hybrid models that combine Neural Networks with other techniques like runtime validation and vulnerability simulation could offer more comprehensive protection. Monitoring emerging SQL injection tactics will ensure the model remains up-to-date and effective. Lastly, the application

Received: 26-05-2025 / Accepted: 25-06-2025 / Published: 28-09-2025

https://doi.org/10.70118/ujet.2025.0202.34

of federated learning could enable secure, distributed model training without compromising sensitive data, paving the way for more robust, scalable, and privacy-aware solutions in SQL injection detection.

References

Alenezi, M., Nadeem, M., & Asif, R. (2021). SQL injection attacks countermeasures assessments. Indonesian Journal of Electrical Engineering and Computer Science, 21, 1121-1131.

Alwan, Z.S., & Younis, M.F. (2017). Detection and Prevention of SQL Injection Attack: A Survey.

Arasteh, G., Loni, H., & Ghassemian, H. (2024). Feature selection using a binary gray wolf optimization algorithm for SQL injection attack detection. Neural Computing and Applications, 36, 14121–14135. https://doi.org/10.1007/s00521-024-09429-z

Dasari, V. R., Suresh, A., Maheswari, G. V. S., & Kumar, G. S. (2025). A novel generative model based approach for synthetic data generation using VAE, CWGAN-GP and UNet for SQLi attack detection. arXiv. https://arxiv.org/abs/2502.04786

Falor, A., Hirani, M., Vedant, H., Mehta, P., & Krishnan, D. (2021). A Deep Learning Approach for Detection of SQL Injection Attacks Using Convolutional Neural Networks. Proceedings of Data Analytics and Management.

Fang, Y., Peng, J., Liu, L. & Huang, C. (2018). OVSQLI: Detection of SQL Injection Behaviors Using Word Vector and LSTM," in Proceedings of the 2nd International Conference on Cryptography, Security and Privacy, pp. 170–174

Farooq, U. (2021). Ensemble Machine Learning Approaches for Detection of SQL Injection Attack. Tehnički glasnik.

Ghosh, P., Saltlake, K., Kolkata, S., Dey, K. & Sengupta, S. (2014). Automatic SQL Query Formation from Natural Language Query. International Conference on Microelectronics, Circuits and Systems (MICRO-2014).

Gu, H., Zhang, J., Liu, T., Hu, M., Zhou, J., Wei, T., & Chen, M. (2020). DIAVA: A Traffic-Based Framework for Detection of SQL Injection Attacks and Vulnerability Analysis of Leaked Data. IEEE Transactions on Reliability, 69, 188-202.

Halfond, W.G.J., Orso, A. (2007). Detection and Prevention of SQL Injection Attacks. In: Christodorescu, M., Jha, S., Maughan, D., Song, D., Wang, C. (eds) Malware Detection. Advances in Information Security, 27. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-44599-1_5.

Hasan, M., Balbahaith, Z., & Tarique, M. (2019). Detection of SQL Injection Attacks: A Machine Learning Approach. 2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA), 1-6.

Invicti(2022). Types of SQL Injection (SQLi). Retrieved from https://www.acunetix.com/websitesecurity/sql-injection2/.

Inyong L., Soonki J., Sangsoo, Y., Jongsub M. (2012). A novel method for SQL injection attack detection based on removing SQL query attribute values.Mathematical and Computer Modelling, 55(1-2), 58 – 68.

Joshi, A., & Geetha, V. (2014). SQL Injection detection using machine learning. 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 1111-1115.

Kaggle (2020). Dataset. Retrieved from https://www.kaggle.com/data sets/sajid576/sql-injection-dataset.

Li, Q., Li, W., Wang, J., & Cheng, M. (2019). A SQL Injection Detection Method Based on Adaptive Deep Forest. IEEE Access, 7, 145385-145394.

Lu, R., Wang, S., & Li, Y. (2021). Research on SQL Injection Detection Model Based on CNN. 2021 International Conference on Intelligent Computing, Automation and Applications (ICAA), 111-114.

Rankothge, W., Randeniya, M., & Samaranayaka, V. (2020). Identification and Mitigation Tool for Sql Injection Attacks (SQLIA). 2020 IEEE 15th International Conference on Industrial and Information Systems (ICIIS), 591-595.

Sheykhkanloo, N. M. (2015). SQL-IDS: evaluation of SQL attack detection and classification based on machine learning techniques," in Proceedings of the 8thInternational Conference on Security of Information and networks. ACM, 258–266.

Sheykhkanloo, N. M. (2017). A learning-based neural network model for the detection and classification of SQL injection attacks. International Journal of Cyber Warfare and Terrorism (IJCWT), 7(2), 16–41.

Singh, N., Dayal, M., Raw, R.S., & Kumar, S. (2016). SQL injection: Types, methodology, attack queries and prevention. 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), 2872-2876.

Tian, Z., Luo, C., Qiu, J., Du, X., & Guizani, M. (2020). A Distributed Deep Learning System for Web Attack Detection on Edge Devices. IEEE Transactions on Industrial Informatics, 16, 1963-1971.

Triloka, J., Hartono, H., & Sutedi, S. (2022). Detection of SQL Injection Attack Using Machine Learning Based On Natural Language Processing. International Journal of Artificial Intelligence Research.

Received: 26-05-2025 / Accepted: 25-06-2025 / Published: 28-09-2025

https://doi.org/10.70118/ujet.2025.0202.34

Zhang, W., Li, Y., Li, X., Shao, M., Mi, Y., Zhang, H., & Zhi, G. (2022). Deep Neural Network-Based SQL Injection Detection Method. Security and Communication Networks. https://www.hindawi.com/journals/scn/2022/4836289/.

Zhuo, Z. Cai, T., Zhang, X. & Lv, F. (2021). Long short-term memory on abstract syntax tree for SQL injection detection, IET Software, 15(2), 188–197.

Received: 26-05-2025 / Accepted: 25-06-2025 / Published: 28-09-2025

https://doi.org/10.70118/ujet.2025.0202.34